New papers Past issues Index of papers

Journal of Digital information, volume 1 issue 4
Themes: Hypermedia systems
1999-01-14

Supporting Software Development in Virtual Enterprises

John Noll and Walt Scacchi
ATRIUM Laboratory, University of Southern California
Email: jnoll@carbon.cudenver.edu, wscacchi@rcf.usc.edu

Key features References; Figures 1, 2, 3, 4, 5, 6; Tables 1, 2, 3

Abstract

This paper presents recent developments in a distributed semantic hypertext framework called DHT that supports software development projects within virtual enterprises. We show how hypertext functionality embodied in DHT solves the practical problems of project coordination. These include collaborative data sharing in a virtual enterprise of distributed teams, integrating existing tools and environments, and enacting software processes to coordinate development activities for teams across wide-area networks.In particular, we describe how software process enactment can be achieved within a virtual enterprise without centralized mechanisms. This is when the process description is represented as a user-navigable hypertext graph the nodes of which associate process steps, staff roles and associated tools with designated software products. Overall, these capabilities provide support for coordinating software development projects across a virtual enterprise of teams connected via the Internet.

Keywords: project coordination, distributed hypertext, software process enactment, tool integration, Internet

Contents

- 1 Introduction
- 2 Overview of the DHT approach
  - Architecture
  - Data model
- 3 Tool integration
  - Object caching
- 4 Incorporating process enactment
- 5 Related work
- 6 Discussion and conclusions
- Acknowledgements
- References

1 Introduction

Software development projects in the future will increasingly take place in an environment where everything is potentially distributed. [S91, N94] The addition of new software project technologies such as agents, architectural middleware, applets, plug-ins, multi-user dialogue systems (MUDs/MOOs), and the World-Wide Web further reinforce the trend toward 'distributed everything'. Software development teams can thus be highly decentralized, both physically and organizationally. Software system products will be produced by loosely coupled 'virtual enterprises' composed of development teams from different organizations who collaborate on specific projects across an information infrastructure, then disband to form new alliances for other projects. Participants in virtual enterprises retain a high degree of autonomy over their own development activities, product data, tools and environments.

These conditions will increase concerns for how to accommodate heterogeneity while maintaining administrative autonomy and transparent access to shared online resources. In addition, participating teams will need to follow well-defined processes to coordinate their work and share objects, resources, intermediate work products--or

more simply, artifacts--with other members. Thus, coordinating software development projects in the face of total distribution requires the ability to access, integrate, communicate and update software products, processes, staff roles, tools and repositories, via a wide-area information infrastructure. [S91, SM97]

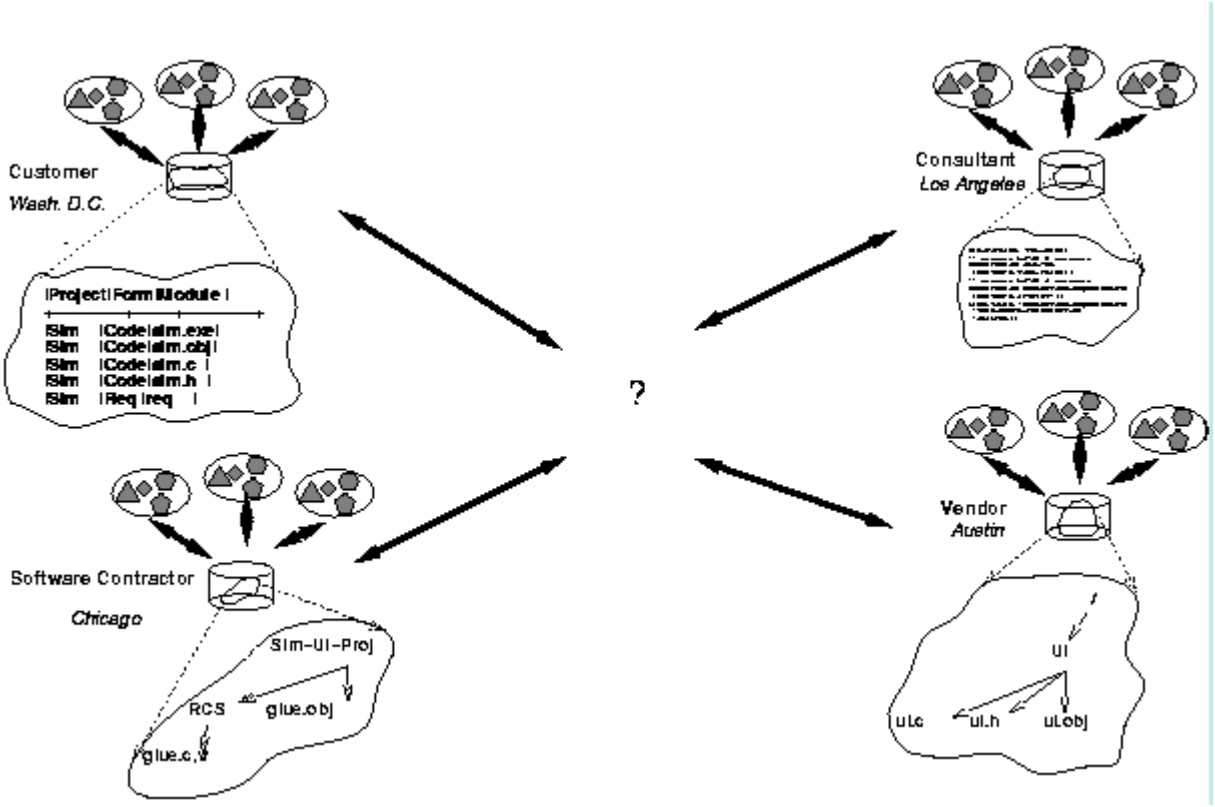

Figure 1. Virtual enterprise scenario for a distributed software development project

Figure 1 depicts an example of such a future development scenario: a loose collaboration among customer, consultant, vendor and software contractor organizations is formed to enhance the customer's legacy simulation system. In this case the contractor will integrate the commercial vendor's user interface software components with the customer's legacy system, according to the process specified by the consultant. Further, the contractor must access, reference and link the customer's requirements specification to corresponding components in the integrated version of the legacy system, the vendor's user interface components, and the contractor added 'glue code' (or middleware), to support requirements traceability, as specified by the consultant's process. Each team has its own tools, software artifacts and repositories (e.g. the customer has a relational database management system as its (SQL/RDBMS) repository, the vendor has a file system repository, the contractor uses a version control (a Unix-based revision control system--RCS) repository, and the consultant has a knowledge-based process model repository). Each team also requires access to at least part of the others' artifacts. Subsequently, each will need to update and expand the shared set of products that represents the project's collective output and deliverables.

Due to geographic separation and the virtual enterprise's loose collaborative structure, each team will be highly autonomous, managing its own computing environment, tools, data and staff. Thus, we cannot assume there will be a single physically central file system, database, agenda, or some other coordination mechanism that is sufficient to serve as the single integrating resource. Similarly, we cannot assume that all teams will agree to

adopt and use any one team's object storage manager, data model, or update control policies. Nonetheless, it will be necessary for each team to be able to modify jointly developed software project artifacts, documents, agendas or process models.

The virtual enterprise needs to produce a set of shared artifacts in some coordinated way, but cannot rely on a central mechanism to provide a shared information space, support communication or coordinate access to shared objects. Thus, the problem denoted by the question mark in Figure 1 and addressed by this paper is:

> *how to coordinate the information objects, tools, work processes, and collaboration among autonomous, decentralized development teams, given the absence of a shared central repository or coordination mechanism in an Internet environment.*

A solution must support:

- Collaboration and information sharing:
  - provide transparent access to artifacts and relationships in heterogeneous, autonomous legacy repositories [NS91]
  - preserve the local autonomy of integrated data, tool or product repositories [NS94]
  - integrate existing/new tools, components and engineering environments in a simple, incremental manner
- Coordination of concurrent activities:
  - integrate the modeling and execution of development processes for different staff roles with specified tools, artifacts and workspaces [MS92]
  - control concurrent access to shared objects to ensure their consistency [NS94]
  - support software process modeling and enactment in widely distributed Internet based environments
- Cooperative communication:
  - provide a clearinghouse of information such as process enactment histories, design discussions, product annotations, etc., that capture and record interaction between developers [GS89, NS91, NS94]
- Composition and configuration:
  - model software development artifacts, documents, versions, staff roles and other relationships [GS89, NS91, NS94, MS96]
  - Support the well-formed composition and interfacing of objects that represent project products, processes, tools, staff roles and supporting environments [MS96, SM97]

In addition, the solution should yield a simple implementation strategy. This goal stems from the purpose of virtual enterprises as a way to react to rapidly changing marketplaces. The cost or effort required of an implementation should not defeat this purpose.

This paper focusses on describing hypertext functionality that addresses three inter-related goals for coordinating software development projects across Internet based virtual enterprises:

1. integrating existing/new tools and environments needed to configure collaborative information-sharing workspaces;
2. supporting software process modeling and enactment to coordinate the tools, product data and cooperative teamwork activities of people working in wide-area settings;
3. providing a simple implementation strategy that operates over intranets or the Internet.

Previous work [NS91, NS94, NS97, SM97] discusses how we address the other goals. Below, we compare our approach with other efforts addressing software development and project coordination over intranets or the

Internet supported by hypertext functionality. Finally, we note that the version of the processing environment described in this paper has been employed to support the development of process-driven software applications that span multiple organizations connected by the Internet. [cf. SN97]

The remainder of this paper is organized as follows. The next section presents an overview of the approach, which centers on the use of a distributed semantic hypertext representation and processing environment we call DHT. This leads into a description of DHT's data model and environment architecture. We discuss the DHT approach to tool integration, and present an approach to software process modeling and enactment using DHT-based hypertext browsing. We conclude with a discussion of related research.

2 Overview of the DHT approach

Our approach to project coordination and sharing of project artifacts is implemented in a framework that employs two complementary forms of information integration:

- *logical* integration provides a view of the shared information space based on a virtual central artifact repository to facilitate project coordination
- *physical* integration provides transparent access to objects that appear in the virtual repository, but are actually stored and managed in autonomous, distributed, heterogeneous repositories.

The structure of the virtual repository is described with a *semantic hypertext* data model. [NS91] Hypertext is an information management concept that organizes data into content objects called nodes, containing text, graphics, binary data, or possibly audio and video, that are connected by links which establish relationships between nodes or sub-parts of nodes. The resulting directed graph, called a hypertext *corpus*, forms a semantic network-like structure that can capture rich data organization concepts while at the same time providing intuitive user interaction via navigational browsing.

The DHT hypertext data model augments the traditional node and link model with aggregate constructs called *contexts* that represent sub-graphs of the hypertext, and *dynamic links* that allow the relationships among nodes to evolve automatically as artifacts are created and modified. The DHT data model defines the structure of objects in the global hypertext, and the operations (including updates) that may be performed on them.

DHT achieves physical integration with a client-broker-server architecture that provides transparent access to heterogeneous repositories through intermediary information brokers we call *transformers*. Clients are software tools (or engineering environments) that developers use to access objects concurrently in server repositories.

Over the past five years that the DHT prototype has evolved, about a dozen different types of software development tools and heterogeneous software repositories have been integrated to run within DHT. [NS91, NS94]

2.1 Architecture

The DHT architecture is based on a client-broker-server model. [ACDC96] Clients implement all application functionality that is not directly involved in a server's storage management. Thus, a client is typically an individual tool, but may be a complete software development environment.

Software artifacts are exported from their native repository through server transformers. A transformer is a kind

of mediator that exports local objects (artifacts and relationships) as DHT nodes and links, and translates DHT messages into local repository operations (Figure 2). Transformers run at the repository site, typically on the same host as the repository; thus, from the repository viewpoint, the transformer appears to be just another local tool or application.

A request-response style communication protocol implements the operations specified in the DHT data model, [NS91, NS94] and includes provisions for locating transformers and authenticating and encrypting messages. The protocol also provides a form of time stamp-based concurrency control [KS86, NS94] to track and prevent 'lost updates'.

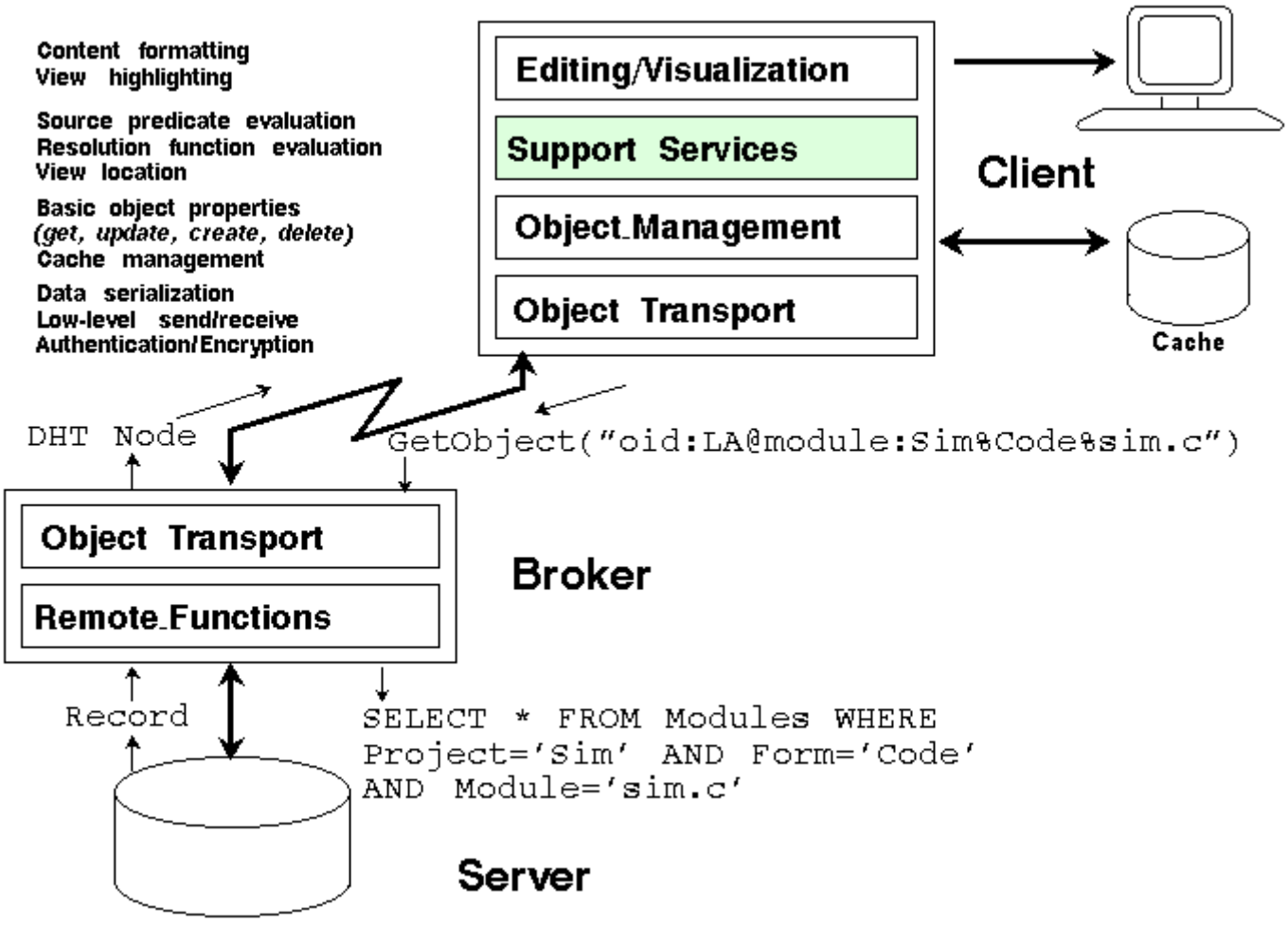

Figure 2. DHT architecture for integrating an SQL/RDBMS repository

Our experience has been that transformers for new repositories can be developed with modest effort (i.e. hours to days), based on reusable server templates that are augmented with code to interact with specific repositories.

2.2 Data model

The DHT data model consists of three types of primitive objects: *nodes*, that represent content objects such as program modules or project documents; *links* that model relationships among nodes; and *contexts*, that enumerate sets of links to allow specification of object compositions as sub-graphs. Nodes, links and contexts are all first class objects having types, attributes and unique object identifiers (OIDs) associated with each. In addition, links have *anchors*, that specify regions or sub-components within node contents to which the endpoints of a link are attached.

Contexts enumerate, but do not actually contain, links. Thus, a given link can be a member of several contexts, making it possible to compose different views of the same objects by imposing different structures or configurations as described by links among those objects. Contexts are also first class objects, and so may serve as the endpoints of links.

A fixed set of operations can be applied to DHT objects: *create*, *delete*, *read* and *update* an object. The owners or administrators of a given repository can elect to provide any subset of these operations (e.g. segmented by user groups, network location, or by type of client), as appropriate for the level of access they intend to offer. In addition, any operation can be performed by a single repository on its own objects. Cooperation among repositories is not required. Therefore, the DHT model preserves a high degree of local repository autonomy.

3 Tool integration

Whether artifacts are stored in a real or virtual repository, software developers create, manipulate and configure shared artifacts using software tools and environments. Many of these objects will exist before the virtual enterprise is formed, and thus before integration by DHT. It is impractical to expect developers and organizations to abandon their favorite tools to use new tools that can access a DHT corpus. Therefore, DHT includes a strategy for migrating existing and new tools to the DHT environment, and a configurable cache mechanism to enable alternative approaches for creating access to, and controlling concurrent updates to, collaborative information spaces.

The migration strategy specifies five levels of integration:

- **Level 0.** At the foundation level DHT provides a process integration capability [cf. MS92] that enables the configuration (via incremental modeling) and binding of individual developers to development roles, process tasks, and product components to appropriate (client-side) tools. During process prototyping the choice of tool(s) may be unspecified (no tool) or specified by class name or similar place holder (tool stub or bitmap), which enables process walkthrough or simulation. [S96, SM97] To support process enactment, executable tools must be bound to corresponding task steps in order to be invoked on the specified product component.
- **Level 1.** At this level tools are not integrated at all. They exist unmodified, or as 'helper applications', and require auxiliary tools (e.g. Web browsers) to interact with DHT on their behalf. Auxiliary tools simply perform node retrieval and update, and link resolution, to and from a tool's standard input/output, or files in the local file system. The use of a Web form-based interface to an existing relational data base management system would be an example.
- **Level 2.** Integration at this level treats DHT nodes as file-like objects. Tools use file system calls like *open (), read(), write()*, etc., to access a node's contents, passing a string representation of the node's OID rather than a file pathname. Level 2 integration can be accomplished without recompiling or modifying source code; simply relinking the tool with a DHT compatibility library (described below) is all that is required. Note, however, that Level 2 tools do not have knowledge of DHT links.
- **Level 3.** At this level a tool is aware of links as relationships among objects, and can follow them. This awareness does not appear at the user interface. An example of a level 3 tool is a document compiler that

resolves links of type 'include' to incorporate text from other nodes into a source node.

- **Level 4.** Last, at this level a tool integrates hypertext browsing and linking into its user interface. This may require extensive modification to the tool's source code. Fortunately, many tools and environments incorporate extension languages or escapes to external programs that can be used to implement linking without re-compilation. For example, this technique was used to implement the DHT editor using GNU Emacs Lisp. Process modeling and enactment are supported at Level 0, and this is described later. It can be used together with any other level of tool integration.

Levels 0 and 1 provide 'facade-level' integration of tools at the user interface. Levels 2 to 4 provide increasing scope for data integration capabilities. Control integration of the kind represented by the use of a software bus or similar message/event broadcast mechanism are not provided, however. As Reiss [R96, p. 405] observes, control integration forces tools to have a common reference framework, which is typically a file name and line number. In this regard, the Level 2 integration scheme for file system emulation could therefore be used to support compatibility with such a control integration scheme. The following sub-sections expand on the role of DHT's file system emulation scheme and object caching framework.
A vast legacy of software development tools and environments use the file system as their repository. These applications read and write objects as files through the file system interface, typically by calling standard I/O library routines supplied for the tool's implementation language. Our goal to provide a reasonable cost implementation strategy precludes requiring that all of these tools be modified to use the DHT tool/application interface in place of the file system library.

To solve this problem, the DHT architecture exploits the file-like nature of DHT atomic nodes to provide a file system emulation library. This library intercepts file system calls and converts them to DHT access operations when strings encoding DHT object identifiers are passed as the pathname argument. As an example, the entry points for the Unix version of this library are shown in Table 1.

Table 1. DHT file system emulation functions

System call	Equivalent DHT operation
open()	Read
access()	same as open()
read()	read() from contents file
write()	write() to contents file
close()	Update; Sync
stat()	stat() on contents file

To enable a tool for DHT access, one simply re-links it with the emulation library. Thus, the tool will continue to function as before when invoked with real file names, yet will retrieve contents from the DHT object cache (described below) when DHT object identifiers are used.

3.1 Object caching

Many software development artifacts which DHT manages change slowly, while others see frequent access during a short period of time. To facilitate collaborative information sharing, and to improve access latency and reduce transformer loads, we have found it desirable to cache frequently used objects, especially those from repositories accessed over the Internet.

A cache layer is built into the the basic request interface to provide transparent node and link caching. The cache is maintained in the local file system; node contents are cached in separate files to support the file system emulation library discussed above, while links and node attributes are cached in a hash table. Clients call a set of object access functions to retrieve objects through the cache layer; these are listed in Table 2.

Table 2. Cache layer interface

Function	Description
Read	Retrieve the specified objects
ReadContents	Return the file containing the object's contents
Update	Update the cached copy of an object
Sync	Update the specified object at the transformer
Source	Evaluate a link's source predicate on a node
DhtResolve()	Resolve a link given a source node and anchor

Each DHT object has a 'time-to-live' attribute that specifies the length of time an object in the cache is expected to be valid. The cache layer uses this attribute, set by the transformer when the object is retrieved, to invalidate objects in the cache upon access. An administrative function periodically sweeps through the entire cache to remove objects whose time-to-live has expired.

The time-to-live attribute is not a guarantee of validity, however. Certain shared objects may be updated frequently by multiple clients. To allow such clients to verify that requested objects have not been modified by another client, the cache layer can be configured with different cache policies to support specific tool/application needs:

- Never use the cached copy; always retrieve an object from the repository.
- Use the cached copy if its time-to-live has not expired.
- Use the cached copy if it has not been modified; verify this by retrieving the object's time stamp from the repository.
- Always use the cached copy if present, regardless of its time-to-live or time stamp.

The cache interface layer does not automatically write updates through to the repository. Instead, a separate function *DhtSync()* causes the cache to send an update request to synchronize the cached copy with that in the repository. This enables DHT integrated tools to tailor access to the cache for different policies for concurrent object access/update. This is especially important when dealing with legacy repositories for software development that impose different user workspace models. Therefore, when using DHT, we need not endorse some particular workspace model as 'best in all circumstances' and thus we can avoid or mitigate some of the costs of transitioning to a different workspace model.

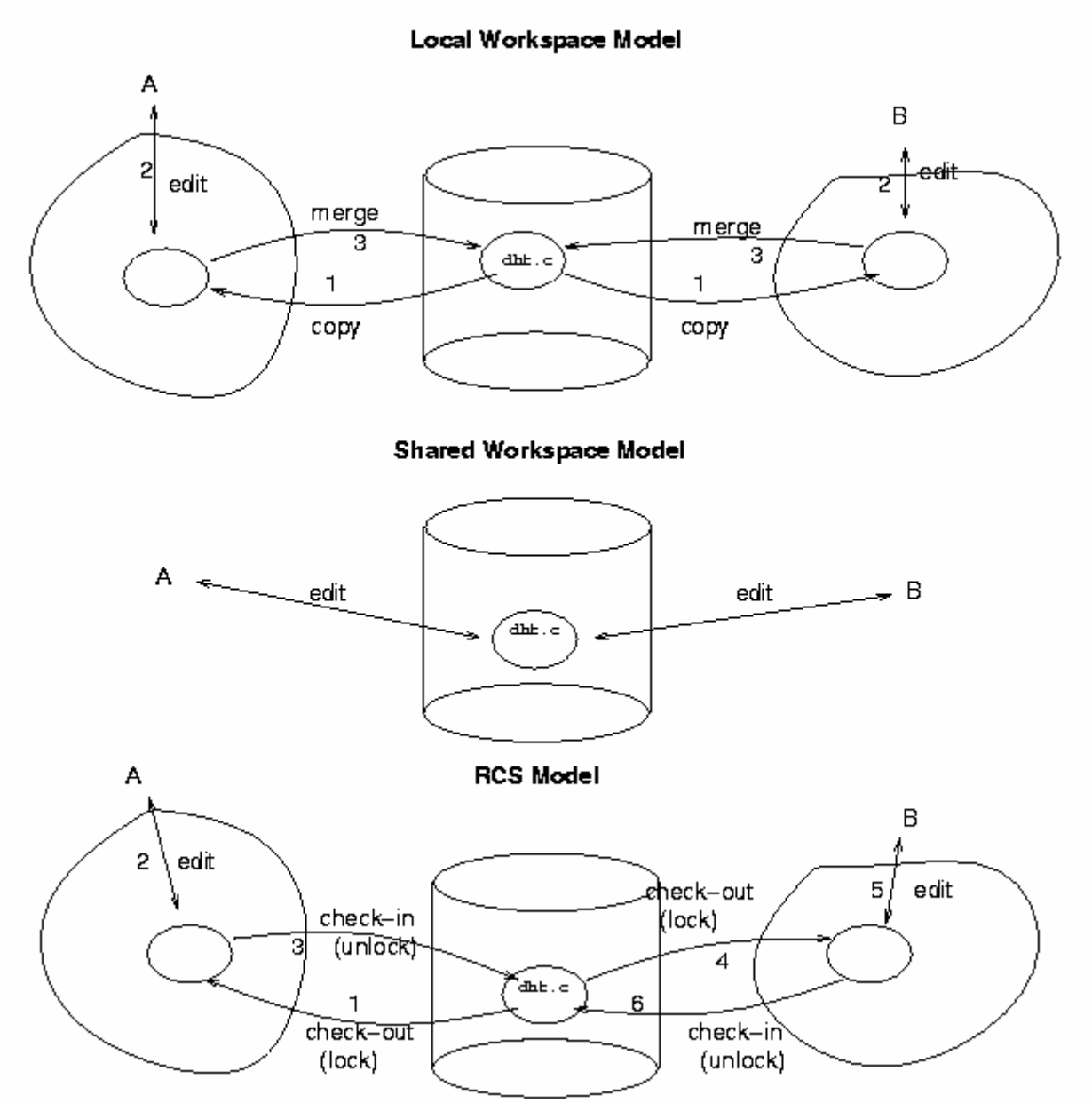

Figure 3. Software development workspace models

As indicated in Figure 3, by delaying synchronization and specifying the non-validating cache policy, the cache can be used as a 'local workspace'. Objects, once placed into the cache, are read and updated locally, and thus are not affected by updates from other developers. A 'sweep' application periodically synchronizes the cached copies, possibly invoking tools that will merge objects that have changed in the interim.

Alternately, updates can be written-through immediately, by calling *DhtSync()* after each update operation. [cf. BHP92] This, coupled with the verifying cache policy, can be used to implement a 'shared workspace' policy for development (Figure 3), in which each developer sees updates from other developers upon object access.

To simulate a 'RCS-style' of version controlled development, in which developers obtain a transaction or

exclusive write access to an object through locking, a *lock* attribute must be added to objects. The lock is bound to the user-ID of the developer who seeks to control the object. The DHT concurrency mechanism ensures that only one developer can set the lock, which is cleared when the object is 'released'. However, applications must cooperate by not modifying objects unless they have successfully set the lock attribute; there is no way to enforce the lock by denying updates if someone insists on updating an object. This policy can be coupled with the validating or non-validating cache policy, depending on the preferences of the developer.

Taken together, the multi-level scheme for integrating new/legacy tools, and support for different policies for configuring object sharing workspaces, provides DHT will an ability to configure and coordinate collaborative workspaces within a project. These workspaces can then be accessed concurrently and updated using tools familiar to distributed developers, even though the individual tools and object repositories may either lack such support, or implement different policies for sharing access and synchronizing updates. Nonetheless, the challenge remains of how best to support cache consistency in the light of the need to maintain autonomy conditions.

4 Incorporating process enactment

A software process is a partially ordered set of tasks performed to develop software. A software process *model* is a description of a software process. If the description is sufficiently formalized, it is possible to execute process models for simulation, analysis and *enactment*. Enactment, in turn, is a computer-supported activity involving one or more developers. Developers perform process steps using integrated tools to create, manipulate, update or configure designated software development artifacts, according to the process fragments assigned to the corresponding staff roles.

Software process enactment uses the formal description of a software process to guide, monitor and control the process by having a process interpreter or engine execute a formal process description. The interpreter can perform three functions:

- **Guidance** involves leading developers through the process by issuing prompts or notifications as to what tasks should be performed at a given time.
- **Monitoring** allows managers and developers to assess the current state and progress of the process.
- **Control** means ensuring the process is followed by restricting developer actions to those that conform to the process description.

A process specifies what steps need to be performed to develop products. At any given time, several products may be in development concurrently. Thus, it is important for a process enactment mechanism to be able to keep track of multiple instances of a process simultaneously, and to be able to cope with the interactions among multiple processes executing concurrently.

For example, a software system may have several developers modifying different program modules at the same time. This means that for each module an instance of a software modification process needs to be executed. Furthermore, different modules developed in different organizations, that are part of a common system, may be constrained by different software processes.

A software process model has a natural representation as a hypertext graph. Nodes in the graph represent tasks or process steps, while links specify both the order in which the tasks should be performed and the products on which they should be performed. The resulting nodes, links and contexts can be browsed and followed just like other hypertext graphs. A hypertext-based process model that links tasks to tools, staff roles and products thus provides a means both for sharing information and coordinating development.

A DHT software process model contains three types of links:

- *Process decomposition* links. These model the decomposition of high-level tasks into lower level, smaller tasks, and eventually into primitive actions. [GS89] These are static links that do not change unless the model itself is modified to correct errors or reflect changes in policy.
- Task and action *precedence* links. These specify the order in which tasks should be performed. These are also static links that evolve slowly.
- *Available task* links. These model the relationship between a particular product node and the tasks which should be performed on it at a given time, as specified by the process model. These are dynamic links that change as the model is enacted.

As an example, the following is an informal description of a process fragment for modifying a program module:

- retrieve module
- edit module to implement changes
- compile module
- unit test module ('run' module, observe output, check req-spec)
- if the unit test is successful, create a new version of the module; otherwise
- debug the module and return to step 1.

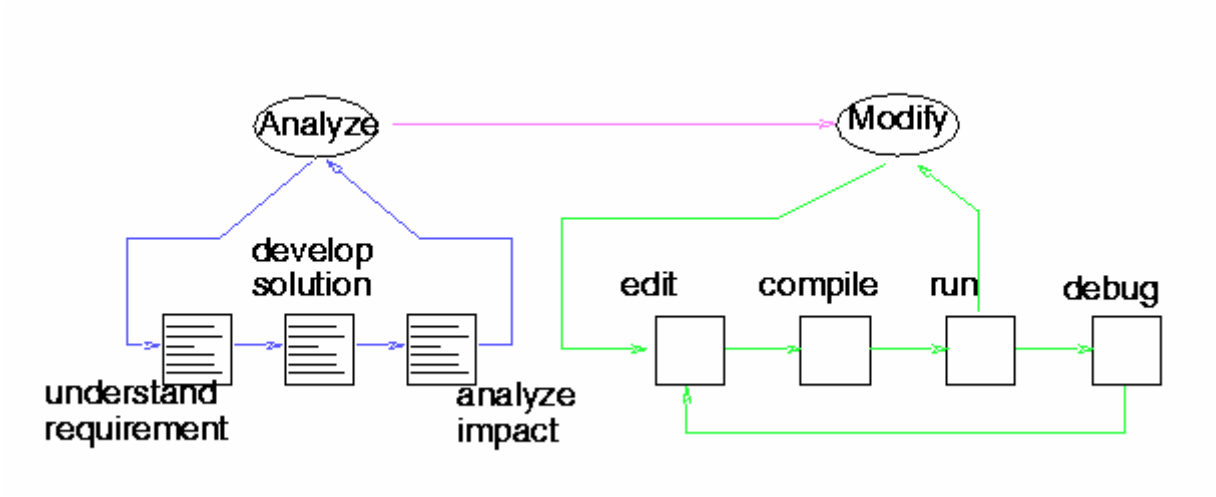

Figure 4. Simple software process model showing decomposition and precedence links

When this description is represented as a DHT process graph (e.g. Figure 4), this model can be instantiated, interpreted and enacted with a *process link server*. A process link server acts as a networked hypertext operating system [NLSS96] that manages user-level processes in a manner somewhat analogous to how a conventional multi-tasking operating system manages computational processes (e.g. resource allocation, context management, maintaining state using program counters, etc.). At a given time, many *instances* of user-level processes may be active, revealing progress made on different software modules by different developers. Each instance has separate process *state* [H92, MS92] including the module being modified, the developer doing the modification, the last step completed, etc. To support process enactment, it is necessary to keep track of this state for each process instance [H92] in order to guide the developer through the process tasks in the appropriate sequence. This is the function of 'available-task' links. Available-task links serve to notify developers that a task should be performed on an artifact by linking the product to a task node.

In enacting a DHT-based process model, developers perform the assigned task/step using the tool(s) integrated and bound to the task/step, accessing and processing the designated software development artifact(s) as specified according to the developer's role. For example, in the software process model in Figure 4, process task flow is indicated by a directed process graph. Suppose the developer is to enact an instance of the 'Modify' process. Figure 5 displays a view of this process. The developer is currently visiting the 'main.c' product, with the 'edit' task pending. The developer performing the edit task creates or updates main.c using the tool 'emacs'. Table 3 shows how the task links are specified for this process.

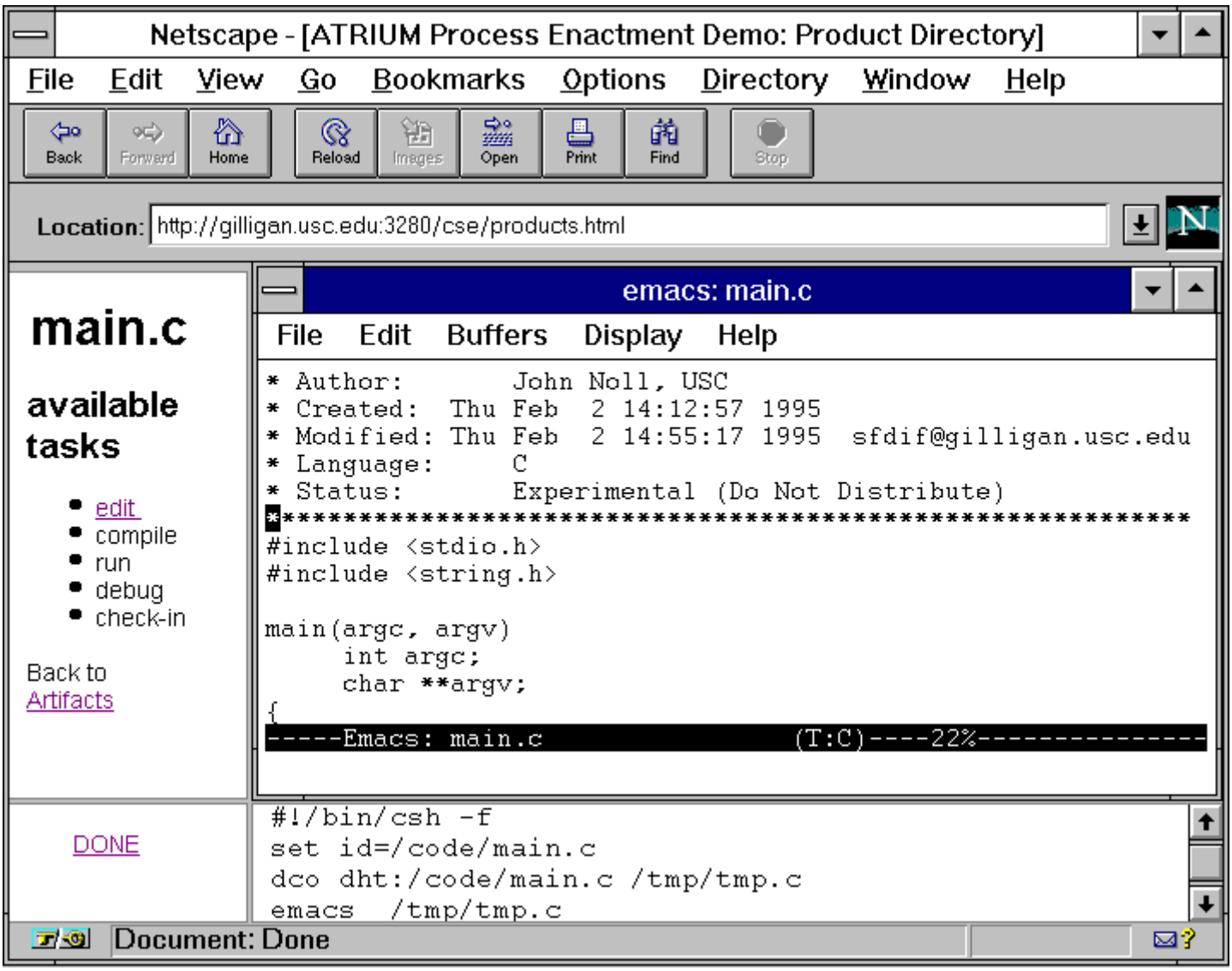

Figure 5. View of a DHT product-centered process user interface

Table 3. Sample of link specifications of type 'DHT:Available-task'

Attribute	Value

source	{[type $node]==DHT:C}
source_anchor	Global
dest	{eval [get-contents process:edit]}
dest_anchor	Global
type	DHT:Available-task

Available-task link for ’edit’ task

Attribute	Value
source	{[type $node]==DHT:C && [status $node]=="edited"}
source_anchor	Global
dest	{eval [get-contents process:compile]}
dest_anchor	Global
type	DHT:Available-task

Available-task link for ’compile’ task

Attribute	Value
source	{[type $node]==DHT:C && [status $node]=="compiled"}
source_anchor	Global
dest	process:unit-test-req-spec
dest_anchor	Global
type	DHT:Available-task

Available-task link for ’run’ task

Process state is represented by available-task links. They reflect the state of the products (artifacts) that the process instance affects: when the ’edit’ task above is performed on a module, its state changes, as reflected by the changes to its contents and time stamp affected by the edit. A link source predicate can examine this state to establish a relationship between a product node and a task node. When the link source predicate is true when applied to the product node, the link will resolve to a task node that should be performed on the product.

In the screen view of Figure 5 the 'edit' link marker indicated in the upper left frame appears as it usually does for Web browsing, whereas the lower right frame displays the contents of the link destination (as task node) as a script. The script is generated automatically by a process model compiler, which also generates the links for the process task steps, product (and product model) and status handler, which are bound to the specified staff role user's workspace. The developer's selection of the 'edit' task thus triggers execution of this script. When this script is activated, the editor is started with the designated software hypertext source code node loaded, as shown in an overlay window in Figure 5. To have external tools that require their own window(s) to appear within the process user interface requires their encapsulation using a display server mechanism such as WinFrame (from Citrix Inc.), WebTermX (from White Pine Inc.), or similar. The lower left frame of Figure 5 indicates the state value ("DONE") of the task that the developer selects when finished with 'edit'. This will then cause the 'available tasks' list to be updated by re-evaluating the source predicates (see Table 3) of process links. The result is that the 'compile' task link is now enabled and ready for selection (as specified by the 'Modify' process model in Figure 4), while 'edit' becomes disabled. The screen view would now unhighlight edit, highlight compile, while the corresponding task link destination would utilize the script for 'compile'. Following the process in Figure 4, the 'run' task takes similar form. After 'run' is completed, then the developer chooses one of 'debug' or 'check-in' tasks to perform, since both are enabled, as suggested in Figure 4.

Task nodes can either be narrative descriptions of the task to be performed, or executable scripts that automatically perform the details of the task. In the latter case, the link's resolution function passes the node to an interpreter to execute the script.

The significance of this approach to modeling process instances is that *the mechanism for process enactment is embedded entirely within the process representation as a hypertext graph*, together with the source predicates and resolution functions of available-task links. In contrast to other enactment systems which employ an environment accessed through a process-based user interface, process-aware tools, or process state databases to execute process specifications, DHT process models can be enacted simply by browsing the process hypertext using any DHT-compatible Web browser or tool. This means that support for wide-area process enactment can be introduced into existing environments with minimal disruption when using DHT.

5 Related work

We have approached integration by providing the illusion of a central repository through the introduction of a layer between storage managers and users of data. Such a layer provides a logically integrated 'virtual' repository of data objects that conform to a semantic hypertext data model. The DHT architecture provides the physical integration of participating repositories necessary for developers to access instances of data objects. This provides a basis for coordinating the processes, artifacts, products, tools and staff roles in a distributed software development project.

Other research uses the same general approach we have taken for providing virtual repository services, but with different data models and results. For example, network file system solutions such as NFS, AFS, or others, [RP93] as well as the Web, implement a common global file system as the integration layer, where each repository exports its objects as files in a single unified directory tree (e.g. via URLs). However, network file systems are a lowest common denominator solution: the directory file model lacks explicit constructs to represent the numerous semantic relationships that exist among software artifacts. [GS89, NS91, MS96] As a result, *ad hoc* techniques such as file naming conventions, and numerous tool-specific databases like Makefiles, tag files, etc., are required to augment the basic directory file model. Consequently, information sharing and project coordination support is at a rudimentary level.

The multi-database approach [BHP92] occupies the other extreme: here the integration layer provides a relational

or object-oriented data model with explicit relationship types. This would seem to solve the relationship problem of the network file systems. However, the complexity of constructing and maintaining a single global schema that captures all of the concepts present in each participating repository, combined with the requirement that integrated repositories have DBMS functionality (query language, data schema, transactions, etc.), generally makes this approach costly and difficult to implement. [FHMS91] Similarly, the reliance on a DBMS for object management services implies the need for a transaction manager to synchronize and coordinate development process events. In contrast, DHT neither assumes nor requires a central database transaction manager or coordination mechanism, yet provides process modeling and enactment, as well as support for heterogeneous object management systems, all within a wide-area information infrastructure. DHT can incorporate both DBMS and non-DBMS repositories. [NS91, NS94] In addition, DHT supports navigational browsing, versioning and multiple workspace models.

A number of research projects have applied hypertext to software object management, including the Hypertext Abstract Machine (HAM), [CG88] the Documents Integration Facility (DIF), [GS89] the Intelligent Software Hypertext System (ISHYS) [GS90] and HyperCASE; [CR92] in fact, DHT's contexts attempt to provide the same abstraction capability as HAM's contexts, while accommodating autonomous repositories. However, these are based on a centralized single repository architecture.

Proxhy, [KL91] Chimera, [ATW94] Endeavors, [BT96] HyperDisco [WIL95] and OzWeb [KDJY97] offer hypertext functionality supporting multiple repositories. In contrast to DHT, Proxhy and Chimera focus on adding new links among existing artifacts. However, existing relationships among the artifacts or data repositories are not translated into links. HyperDisco also addresses tool integration and provides a hierarchy of integration levels. HyperDisco allows linking to artifacts that are not integrated into the hypertext, so tool integration is characterized by the degree to which artifacts bound to a tool can be linked to other nodes. In contrast, DHT assumes links are always between objects exported as hypertext nodes, and thus characterizes tool integration according to the degree to which a tool manipulates and presents links.

The Open Hypermedia Systems Working Group [OHSWG97] is attempting to define standards for interoperability among different hypermedia systems. Among the goals are to incorporate data from various repositories, including non-hypermedia document repositories. In contrast to DHT, however, they do not explicitly address transforming native structure into links; rather, the focus seems to be on adding link support to existing applications and repositories.

Various approaches to linking incorporate different degrees of dynamism. Microcosm, for example, provides 'generic links" that link spans of text found in any node. [LDH92] Ashman and Verbyla propose a "functional model of the link"' to characterize the degree of 'externalization" and dynamism of different linking schemes. [AV94] DHT's dynamic links are an implementation of this model and as such provide both 'full' dynamism and externalization.

ISHYS [GS89] and Trellis [S94] explored software process modeling and enactment via hypertext. SigmaTrellis utilizes a petri net based process enactment formalism, while ISHYS utilizes one based on a semantic network the transitions of which are controlled by rule-based triggers. However, both ISHYS and sigmaTrellis provide process enactment support through a centralized architecture. More recently HOSS, [NLSS96] Endeavors [BT96] and OzWeb [KDJY97] have addressed process support and have extended their hypertext mechanisms to support distributed capabilities similar to DHT, but with some differences. For example, HOSS, Endeavors, OzWeb and DHT all provide a semantic hypertext modeling and process enactment capability, but differ in how process behavior is specified (as methods or rules attached to activity objects for HOSS, Endeavors and OzWeb; as predicates attached to dynamically updated precedence links in DHT).

Endeavors, OzWeb and HOSS address issues of versioning, as does DHT. [NS94] Based on the available reports,

however, these systems have not yet addressed the range of standard configuration management (CM) services, [D91] alternative workspace models and CM update policy models [F91] that have been addressed by DHT. [NS94, NS97]

The OzWeb prototype seeks to provide an overall hypermedia collaboration environment supporting distributed software project teams who collaborate and interact in 'groupspaces' that manipulate 'subwebs' of software artifacts. These groupspaces incorporate concepts from multi-user dialogue systems (MUDs), which represent a new kind of hypertext functionality that can support project communication, information sharing, and a place to find and use tools.

Endeavors, HyperDisco, Desert and DHT provide alternative approaches to tool integration. Desert [R96] was not designed around the use of software hypertext, as were the others. HyperDisco [WIL95] provides an object-oriented scheme for tool integration, but an object-oriented scheme that inherits objects across a network can violate the conditions for autonomy that DHT seeks to maintain. Endeavors has recently been extended [W97] to support the integration of external applications or tools using a Chimera based scheme for Endeavor-to-tool communications and display presentation. [ATW94] Nonetheless, in DHT five levels of tool integration support the kinds of approaches that these three environments individually support.

There is growing interest in developing new technologies to enable to rapid formation and (re)configuration of Internet- based virtual enterprises. Software product development and distribution, [N94] virtual software configuration, [BNT96] and global team sub-contracting [CPC96] are of interest, as are applications to other engineering design and manufacturing domains. [FTS+96] These efforts also envision different scenarios and support mechanisms to facilitate project coordination. Nejmeh [N94] focuses on the inherent potential for the Internet to serve as a medium for coordinating distributed development projects. However, he does not propose a specific design or reference implementation to describe how project coordination might be achieved. Boldyreff *et al*. [BNT96] highlight the need to support software configuration management processes across virtual enterprises as an essential element of distributed project coordination. However, although they do not describe the design or prototyping of such a coordination support environment, much of what DHT provides, as outlined in this and a companion paper, [NS97] moves towards demonstrating the approach they envision. Similarly, one vision from IBM [CPC96] for how to support and coordinate project management when employing globally distributed development sources also proposes using a DHT-like scheme as the underlying project coordination approach for tool, data and process integration.

Elsewhere, we find efforts in advanced manufacturing and concurrent engineering research projects that are exploring the potential of Internet-based agents and multi-agent design environments. For example, efforts at Carnegie-Mellon University [FTS+96] seek to exploit the potential of the Web as a development and manufacturing infrastructure for distributed projects. While providing project coordination services was not explicitly addressed, they propose to use Web-based systems to integrate heterogeneous tools, product data and repositories. In particular, the CMU effort cites the influence of our earlier work on semantic hypertext [GS89] as a factor shaping the design and implementation of their approach. Thus, it appears that DHT-based or DHT-derived capabilities could be brought to bear in supporting project coordination in distributed manufacturing design and engineering projects, as well as in business process reengineering. [SM97, SN97]

6 Discussion and conclusions

We began with the assertion that in the future software development will be performed by cooperating development teams. These teams will be autonomous, widely distributed and loosely associated in virtual enterprises, yet will need to share data in the form of software artifacts, as well as coordinate and configure relationships among them, to order and harmonize the products of software development processes.

Project coordination entails supporting collaborative information sharing and tool usage, concurrent development processes and data updates, intra- and inter-team communication, and well-formed composition of configured software artifacts, products and documents. Software development projects that operate within virtual enterprises cannot assume that a central repository or similar mechanism will exist to serve as the medium of coordination among developers. Thus, the problem we addressed was how can virtual enterprises share, manipulate and update data among their loosely-coupled teams. The solution we described provides a semantic hypertext to serve as a virtual repository and distributed process enactment infrastructure, and thus as a project coordination mechanism for virtual enterprises.

Our solution provides an incremental integration layer between autonomous software repositories and their users. It in turn provides the appearance of a central repository as a coordination mechanism while maintaining the distributed physical environment. [cf. SN97] Figure 6 portrays one view of this. The integration layer achieves three types of integration:

- *logical* integration by describing data and operations on them in terms of a common semantic hypertext data model
- *physical* integration via a distributed architecture for access to autonomous repositories
- *process* integration via representation and enactment of user-level work processes encoded within a semantic hypertext interpreted by a process link server

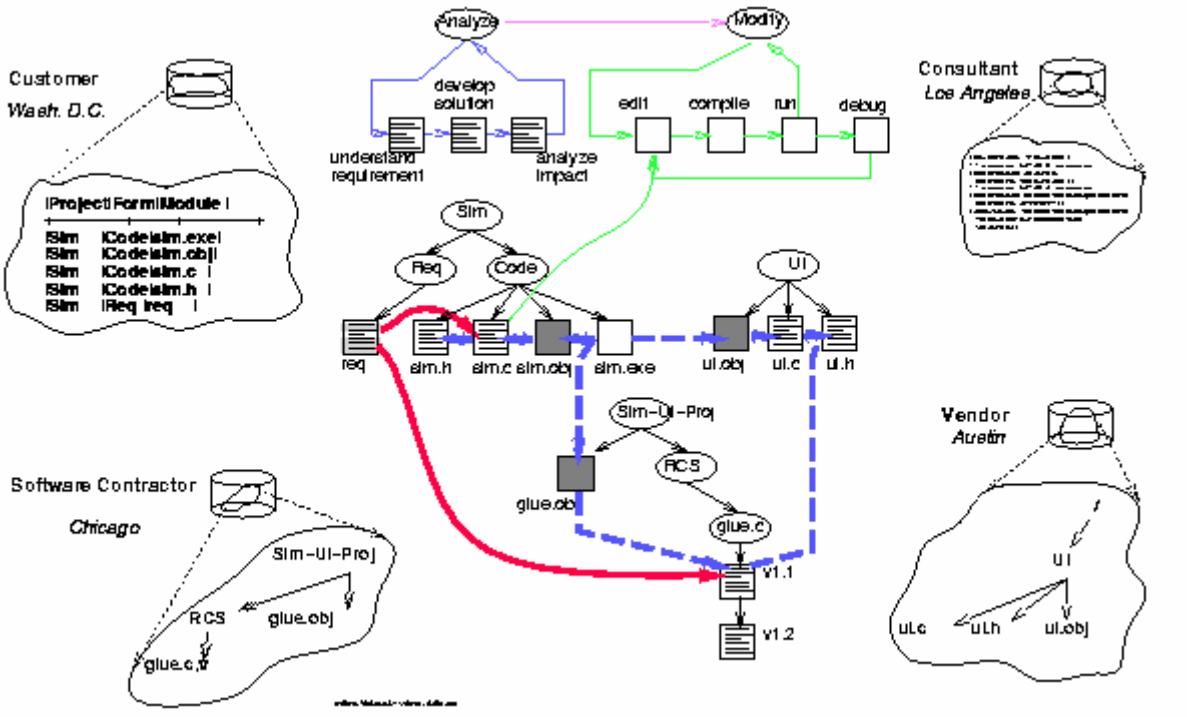

Figure 6. Snapshot of integration in a virtual enterprise via DHT

The DHT approach has the following benefits:

- **Evolutionary approach to coordination and integration.** It is possible to migrate a conventional Unix toolbox-oriented environment to DHT without recompiling any of the individual tools. This is because

DHT offers several levels of tool integration, depending on the degree of 'hypertext awareness' desired for a given tool. This incremental evolutionary approach to incorporating existing tools into a DHT environment enables tool integrators to make trade-offs between integration effort and hypertext functionality. This gives administrators great flexibility to preserve existing investment in tools and training while simultaneously obtaining the advantage of DHT's integration and hypertext capabilities.

- **Supporting project team coordination through transparent process enactment.** By representing software development processes as semantic hypertext graphs, DHT achieves enactment without the need for an explicit process interpreter or environment. This is a significant advantage in a virtual enterprise, where each participant may already have a favorite development environment in place. The DHT approach allows process enactment to co-exist with existing tools and environments. Thus, as hinted at in Figure 5 and suggested by Figure 6, DHT provides support for integrating products, organizations (and staff roles, not shown), tools and processes that may all be distributed across the Internet in a transparent manner.
- **Comprehensive solution.** The most significant contribution of DHT is the way it applies the features of hypertext to data integration, combining intrinsic support for user interaction with data modeling and access. DHT combines the advanced data modeling capabilities of semantic networks with the familiar navigation-based access of file systems, and the intuitive direct manipulation browsing features of hypertext. This means that as soon as a transformer is put in place to export data from a particular repository, the user interface and access operations to those data are also in place. Furthermore, both the user and access interface conform to a single common model, therefore maintaining highly transparent access to heterogeneous data. The result is effective yet low in implementation cost.

We set out to develop a solution to the problem of how to coordinate software development projects within a virtual enterprise. This entails support for:

- collaborative sharing software engineering data among distributed, autonomous development teams
- data modeling and management appropriate for software artifacts and relationships
- transparent access to heterogeneous, autonomous legacy repositories
- multi-level tool integration
- process modeling and wide-area enactment
- low implementation cost.

DHT achieves these goals by applying semantic hypertext concepts and functionality to logical and physical integration. In so doing, DHT solves practical problems of sharing data and coordinating work processes in a virtual enterprise, and establishes a basis for continuing research in integrating heterogeneous software object management repositories, data models and implementation architectures using easily navigated wide-area hypertexts.

Acknowledgements

An earlier version of this paper was presented at the *Hypertext '96 Workshop on Incorporating Hypertext Functionality into Software Systems*, Washington, DC, in February 1996, and at the *1996 California Software Symposium*, USC, Los Angeles, CA, in April 1996. Support for this research and preparation of this report was provided by ONR grant N00014-94-1-0889. No endorsement implied. John Noll contributed to this work while working as a research associate at the USC ATRIUM Laboratory. At present, he is an assistant professor of Computer Science at the University of Colorado in Denver, CO.

References

[ACDC96] **Ashman, H., Cawley,T., Davis, S. and Chase, G.** (1996) "Issues in the Use of External and Remote Services in Hypermedia Systems". *Workshop on Hypertext Functionality (HTFII)*, Washington, DC, March http://www.ep.cs.nott.ac.uk/~hla/HTF/HTFII/Ashman.html

[ATW94] **Anderson, K. M., Taylor, R. N. and Whitehead Jr., E. J.** (1994) "Chimera: Hypertext for heterogeneous software environments". In *Proc. European Conference on Hypermedia Technology*, Edinburgh, Scotland, September, pp. 94-107

[AV94] **Ashman, H. and Verbyla, J.** (1994) "Dynamic Link Management Via the Functional Model of the Link". *Proceedings of the Basque International Workshop on Information Technology*, February http://www.cs.flinders.edu.au/research/hypermedia/biwit94.html

[BHP92] **Bright, M. W., Hurson, A. R. and Pakzad, S. H.** (1992) "A taxonomy and current issues in multidatabase systems". *IEEE Computer*, Vol. 25, No. 3, March, 50-61

[BNT96] **Boldyreff, C., Newman, J. and Taramaa, J.** (1996) "Managing Process Improvement in Virtual Software Corporations". *Proceedings of WET ICE '96 (*IEEE Press)

[BT96] **Bolcer, G. A. and Taylor, R. N.** (1996) "Endeavors: A Process System Integration Infrastructure". *Proceedings of the 4th International Software Process Conference*, Brighton, UK, December

[CG88] **Cambell, B. and Goodman, J.M.** (1998) "HAM: A General Purpose Hypertext Abstract Machine". *Communications of the ACM*, Vol. 31, No. 7, July

[CPC96] **Chaar, J., Paul, S. and Chillarege, R.** (1996) "Virtual Project Management for Software". *Proceedings NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-art and Future Directions*, University of Georgia, Athens, GA, May http://lsdis.cs.uga.edu/activities/NSF-workflow/santanu.html

[CR92] **Cybulski, J. J. and Reed, K.** (1992) "A hypertext based software engineering environment". *IEEE Software*, March

[D91] **Dart, S.** (1991) "Concepts in Configuration Management Systems". In *Proceedings of the Third International Workshop on Software Configuration Management*, ACM Sigsoft, pp. 1-18

[F91] **Feiler, P. H.** (1991) "Configuration Management Models in Commercial Environments". Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie Mellon University, March

[FHMS91] **Fang, D., Hammer, J., McLeod, D. and Si, A.** (1991) "Remote-exchange: An approach to controlled sharing among autonomous, heterogeneous database systems". In *Proceedings IEEE CompCon*, San Francisco, February

[FTS+96] **Finger, S., Terk, M., Subrahmanian, E., Kasabach, C., Prinz, F., Siewiorek, D.P., Smailagic, A., Stivoric, J. and Weiss, L.** (1996) "Rapid Design and Manufacture of Wearable Computers". *Communications of the ACM*, Vol. 39, No. 2, 63-70

[GS89] **Garg, P. K. and Scacchi, W.** (1989) "ISHYS: Designing an intelligent software hypertext system". *IEEE Expert*, Vol. 4, No. 33, Fall, 52--63 (An earlier version appeared in *Proceedings of ACM Hypertext '87 Conference*, Chapel Hill, North Carolina, 1987)

[GS90] **Garg, P.K. and Scacchi, W**. (1990) "A Hypertext System to Manage Software Life Cycle Documents". *IEEE Software*, Vol. 7, No. 3, May, 90-99

[H92] **Heimbigner, D.** (992) "The ProcessWall: A process state server approach to process programming". In *Proceedings of the fifth SIGSOFT Symposium on Software Development Environments*, Tyson's Corner, Virginia, December

[KDJY97] **Kaiser, G.E., Dossick, S.E., Jiang, W. and Yang, J.J.** (1997) "An Architecture for WWW-based Hypercode Environments". *Proceedings of the 19th IEEE International Conference on Software Engineering*, Boston, MA, May

[KL91] **Kacmar, C.J. and Leggett, J.J.** (1991) "PROXHY: A Process Oriented Extensible Hypertext Architecture". *ACM Transactions on Information Systems*, Vol. 9, No. 4, October, 399-419

[KS86] **Korth H.F. and Silbershatz, A.** (1986) *Database System Concepts (*McGraw-Hill)

[LDH92] **Li, Z., Davis, H. and Hall, W.** (1992) "Hypermedia Links and Information Retrieval". Presented at *British Computer Society 14th Information Retrieval Colloquium*, Lancaster, UK, April http://www.mmrg.ecs.soton.ac.uk/publications/archive/li1992/

[MS92] **Mi, P. and Scacchi, W.** (1992) "Process integration in CASE environments". *IEEE Software*, Vol. 9, No.2, March, 45-54 http://www.usc.edu/dept/ATRIUM/Papers/CASE_Process_Integration.ps

[MS96] **Mi, P. and Scacchi, W.** (1996) "A knowledge-based meta-model for formulating models of software development processes". *Decision Support Systems*, Vol. 17, No. 3, 313-330 http://www.usc.edu/dept/ATRIUM/Papers/Process_Meta_Model.ps

[N94] **Nejmeh, B.A.** (1994) "Internet: A Strategic Tool for the Software Enterprise". *Communications of the ACM*, Vol. 37, No. 11, November, 23-27

[NLSS96] **Nurnberg, P.J., Leggett, J.J., Schneider, E.R. and Schnase, J. L.** (1996) "Hypermedia Operating Systems: A New Paradigm for Computing". *Proceedings of ACM Hypertext '96*, Washington, DC, March http://www.cs.unc.edu/~barman/HT96/P23/hossfin.html

[NS91] **Noll, J. and Scacchi, W.** (1991) "Integrating diverse information repositories: A distributed hypertext approach". *IEEE Computer*, Vol. 24, No. 12, December, 38-45 http://www.usc.edu/dept/ATRIUM/Papers/Distributed_Hypertext.ps

[NS94] **Noll, J. and Scacchi, W.** (1994) "A hypertext system for integrating heterogeneous, autonomous software repositories". In *Proceedings of the third Irvine Software Symposium*, University of California, Irvine, CA, April, pp. 49-59 http://www.usc.edu/dept/ATRIUM/Papers/Integrating_Software_Repositories.ps

[NS97] **Noll, J. and Scacchi, W.** (1997) "Supporting Distributed Configuration Management in Virtual Enterprises".
In *Software Configuration Management*, edited by R. Conradi, Lecture Notes in Computer Science, Vol. 1235, pp. 142-160

[OHSWG97] **Open Hypermedia Systems Working Group** (1997) November http://www.csdl.tamu.edu/ohs/ohswg.html

[R96] **Reiss, S.** (1996) "Simplifying Data Integration: The Design of the Desert Software Development Environment". *Proceedings of the 18th IEEE International Conference on Software Engineering*, Berlin, March, pp. 398-407

[RP93] **Rao, H.C. and Peterson, L.L.** (1993) "Accessing files in an Internet: the Jade file system". *IEEE Transactions on Software Engineering*, Vol. 19, No. 6, June, 613-625

[S91] **Scacchi, W.** (1991) "A software infrastructure for a distributed system factory". *IEE Software Engineering Journal*, Vol. 6, No. 5, 355-369

[S96] **Scacchi, W.** (1996) "The life cycle engineering of complex processes and capabilities: An experience report". Interactive presentation http://www.usc.edu/dept/ATRIUM/Software_Process.html

[SM97] **Scacchi, W. and Mi, P.** (1997) "Process Life Cycle Engineering: A Knowledge-Based Approach and Environment".
Intelligent Systems in Accounting, Finance, and Management, Vol. 6, 83-107
http://www.usc.edu/dept/ATRIUM/Papers/Process_Life_Cycle.html

[SN97] **Scacchi, W. and Noll, J.** (1997) "Process-Driven Intranets: Life Cycle Support for Process Reengineering". *IEEE Internet Computing*, Vol. 1, No. 5, September, 42--49
http://pdf.computer.org/ic/books/ic1997/pdf/w5042.pdf

[S94] **Stotts, P.D.** (1994) "Trellis: Process Models as Multi-Reader Collaborative Hyperdocuments." *Proceedings of the 9th Annual Software Process Workshop*, Airlie, VA, October, pp. 85-89

[W97] **Whitehead Jr., E.J.** (1997) "An Architectural Model for Application Integration in Open Hypermedia Environments". *Proceedings of the 8th ACM Hypertext Conference*, Southhampton, UK

[WIL95] **Wiil, U.** (1995) "HyperDisco: An Object-Oriented Hypermedia Framework for Flexible Software System Integration". Proceedings of the 19th Annual International Computer Software and Applications Conference (COMPSAC'95), Dallas, TX, August, pp. 298-305
http://www.daimi.aau.dk/~kock/Publications/HyperDisco/COMPSAC95.ps.gz